Improved TiKV Observability:

How We Trace Events under Nanoseconds Latency

Wish & Zhenchi @ TiKV · KubeCon NA 2020

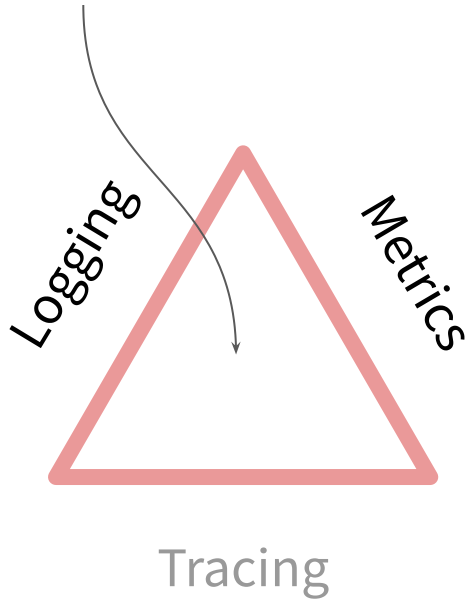
An open source distributed transactional key-value database

What is TiKV?

CLOUD NATIVE COMPUTING FOUNDATION	CNCF Graduated
8K+	GitHub Stars
200+	Contributors

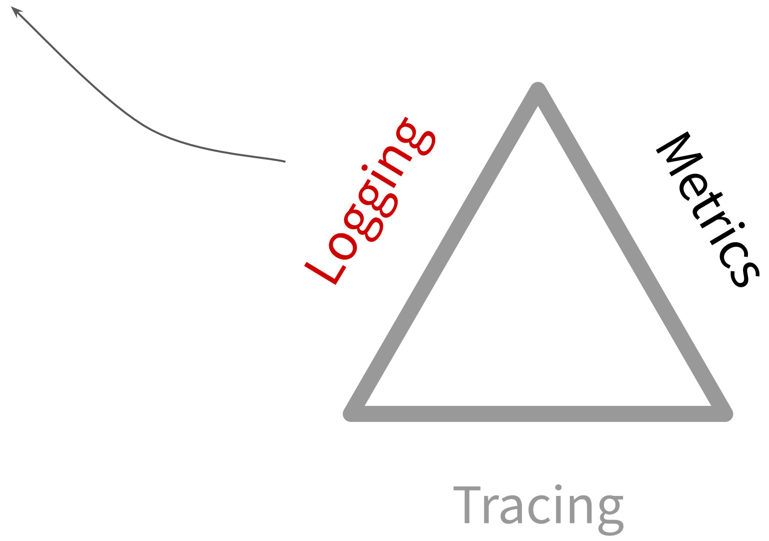
Why Tracing?

Why there is a write jitter?

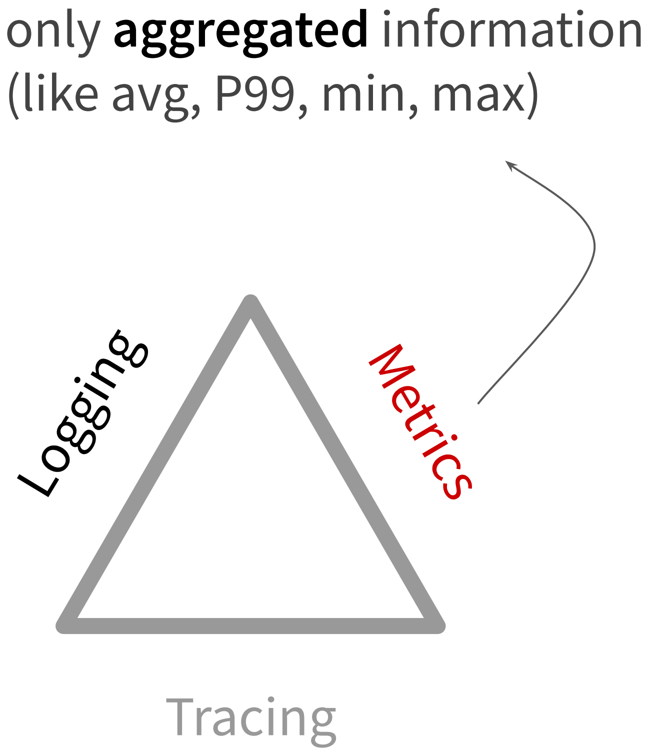


hard to **link** everything related to a request together

Why Tracing?

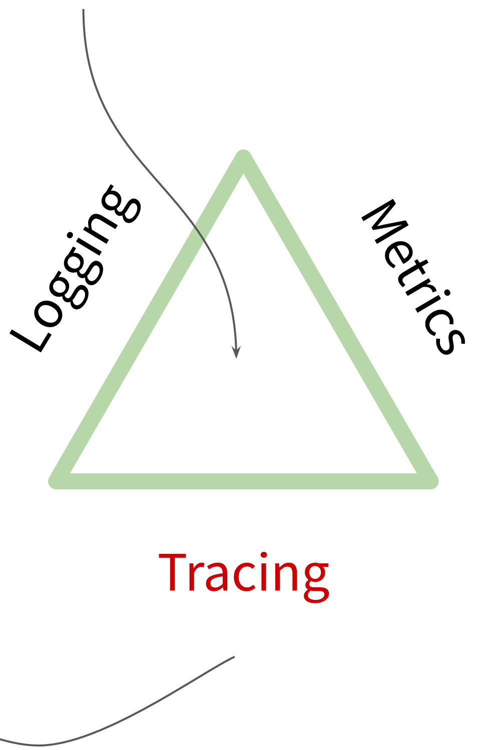


Why Tracing?


only **aggregated** information
(like avg, P99, min, max)
Logging
Metrics
Tracing

Why Tracing?

Why there is a write jitter?



we want to use trace to know it!

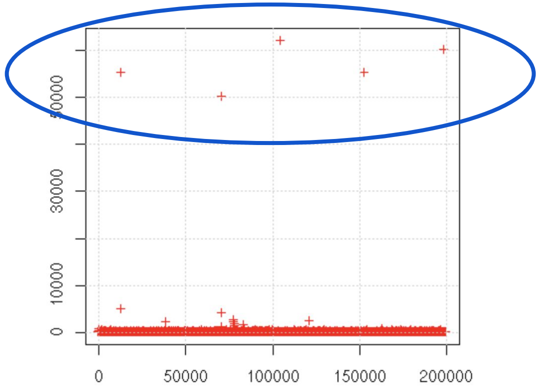
Tracing Library?

OpenTracing / OpenTelemtry compatible in Rust

- Tokio Tracing
  - github.com/tokio-rs/tracing
- Rustracing
  - github.com/sile/rustracing
- ...

Challenge 1/2

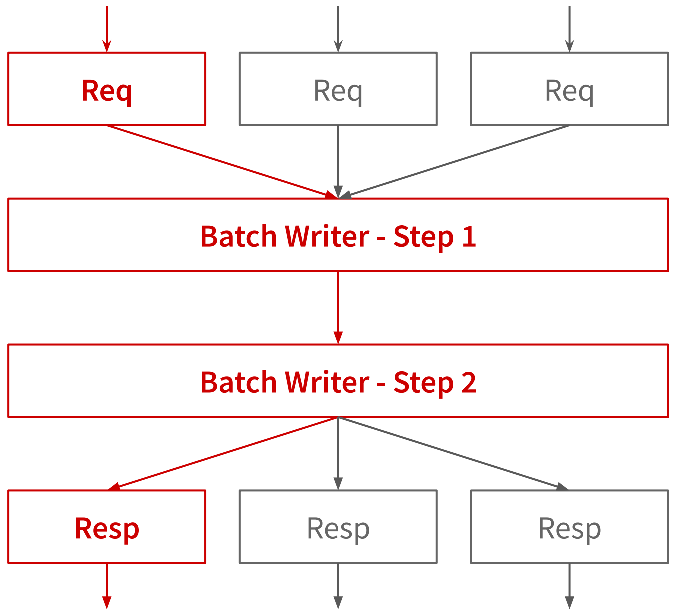
We want to catch ***jitters***



- All KV requests need to be traced.
- Each KV request may only take ~1μs.
- *So tracing must be **super** efficient.*

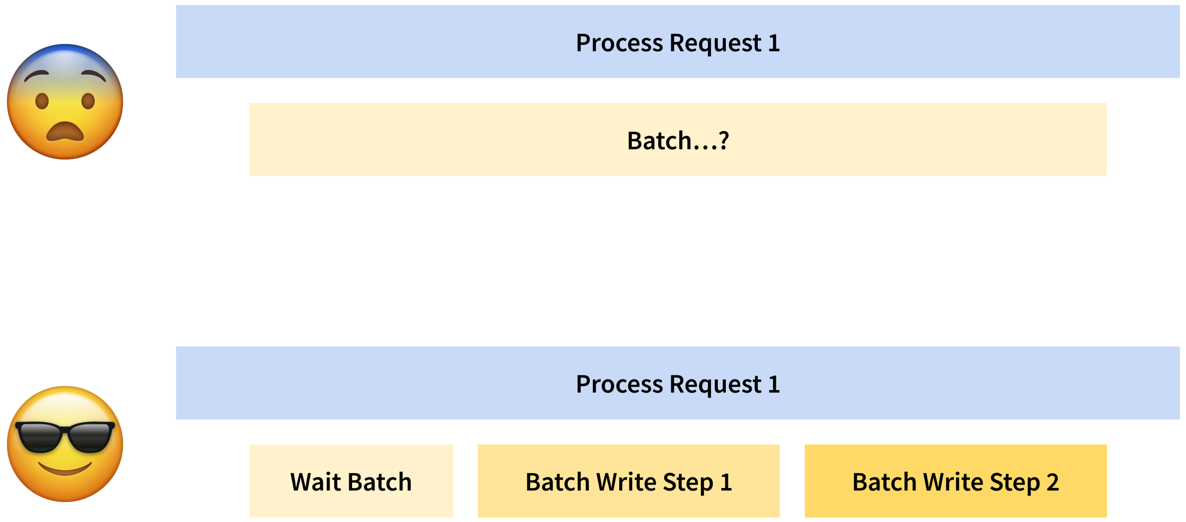
Challenge 2/2

batch processing pattern?



Challenge 2/2

batch processing pattern?



Solution

OpenTracing / OpenTelemtry compatible in Rust

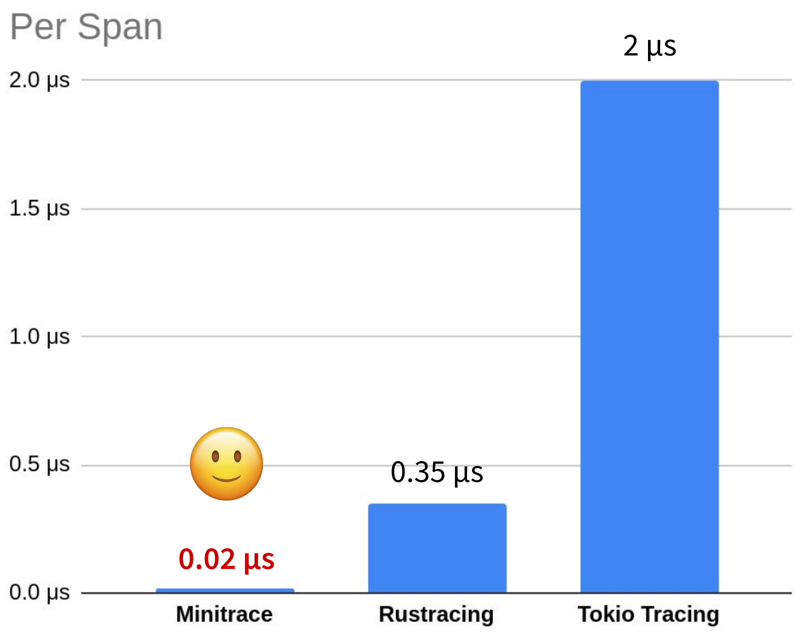
- Tokio Tracing
  - github.com/tokio-rs/tracing
- Rustracing
  - github.com/sile/rustracing
- ...

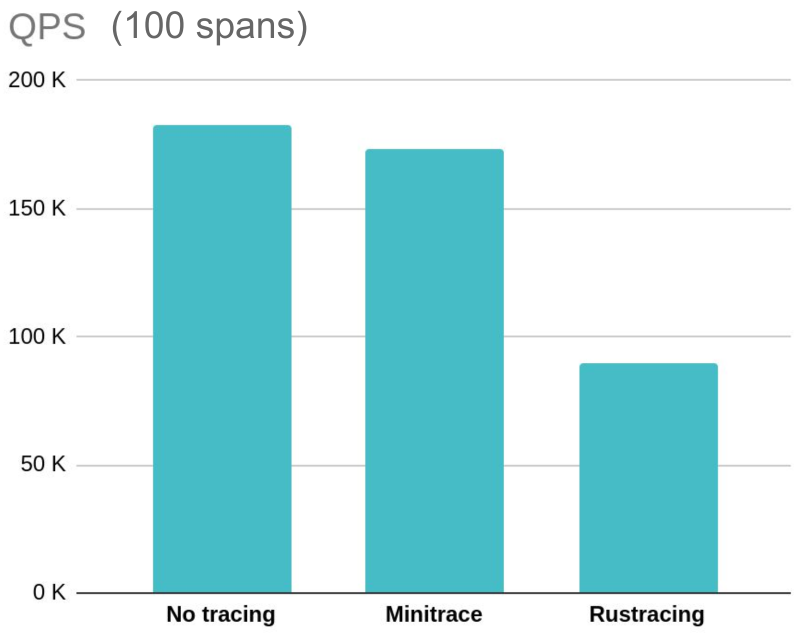
Solution

OpenTracing / OpenTelemtry compatible in Rust

- Tokio Tracing
  - github.com/tokio-rs/tracing
- Rustracing
  - github.com/sile/rustracing
- **POC prototype: minitrace**

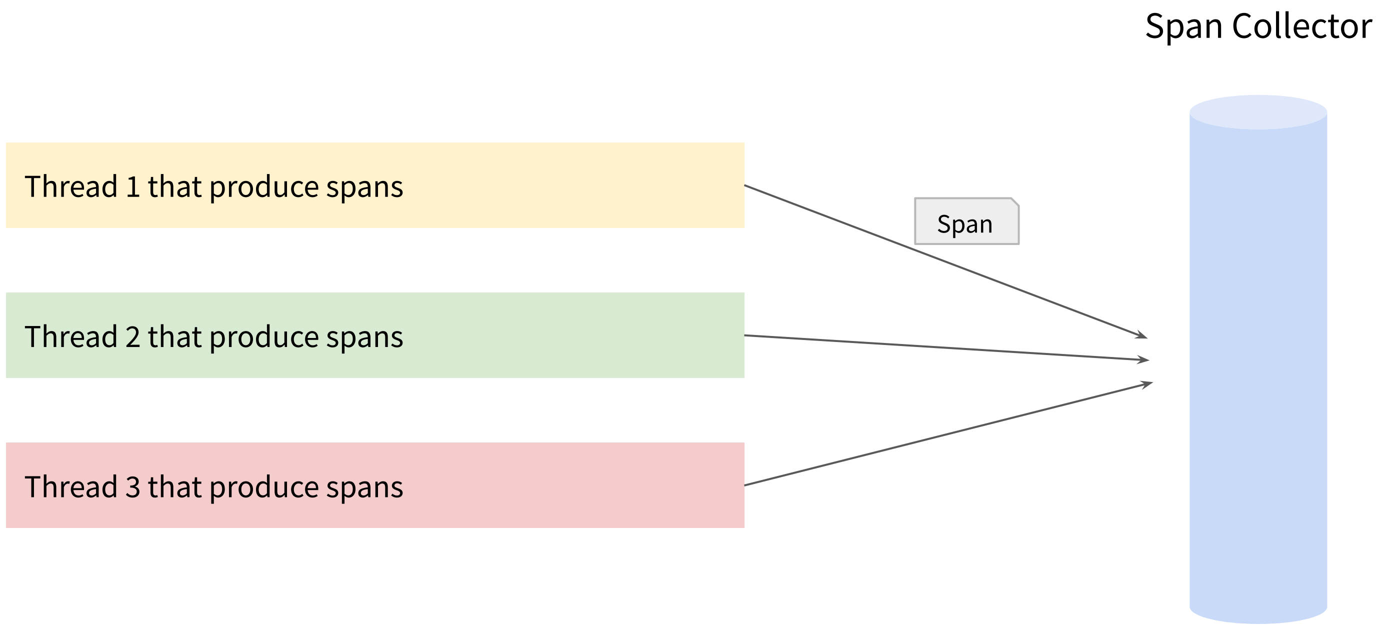
Performance: 20ns/span





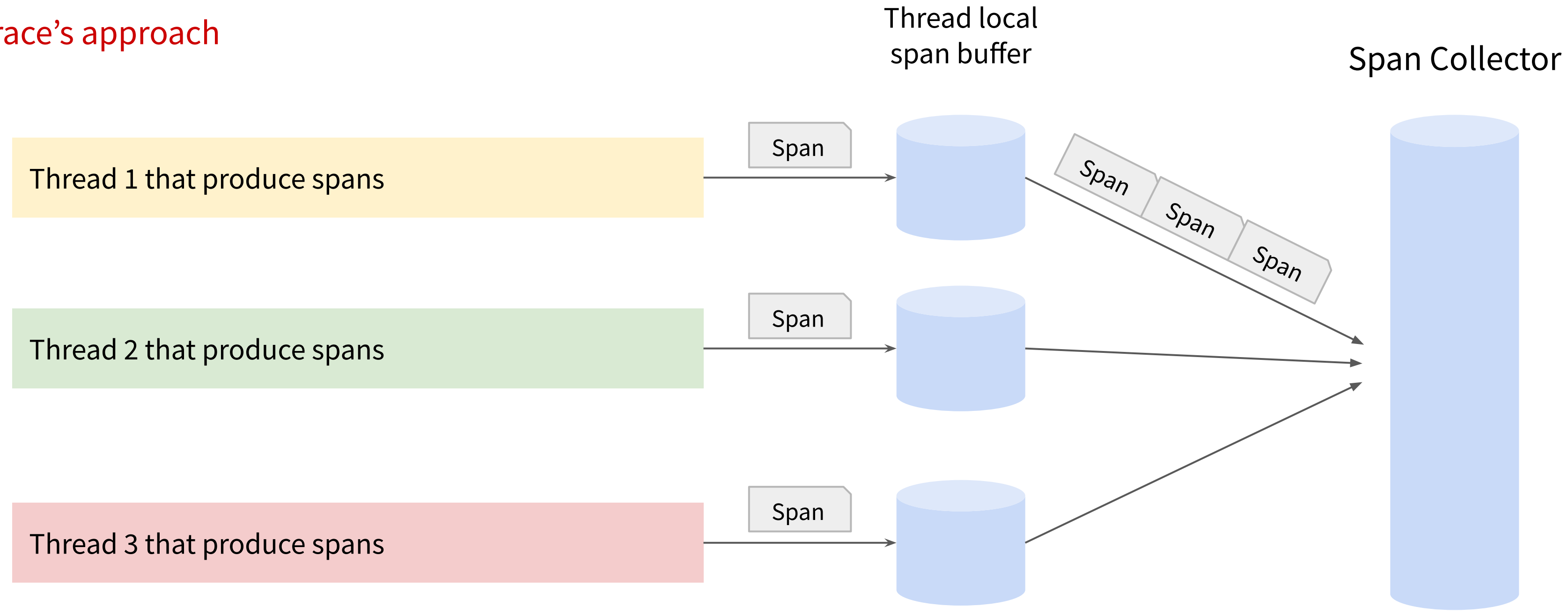
Performance: Reduce Contention

Common pattern


Span Collector
Thread 1 that produce spans
Thread 2 that produce spans
Thread 3 that produce spans
Span

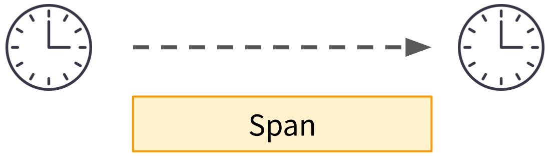
Performance: Reduce Contention

Minitrace's approach


Thread local span buffer
Span Collector
Thread 1 that produce spans
Thread 2 that produce spans
Thread 3 that produce spans
Span
Span
Span
Span
Span
Span

Performance: Faster Timing

- A basic span: when it is started, when it is ended.



- CLOCK_MONOTONIC (with vDSO)?

  × 25 ns ………. 16% overhead in KvGet with 10 spans

- CLOCK_MONOTONIC_COARSE?

  √ 5 ns ………. 3% overhead in KvGet with 10 spans

  × Precision can be as low as 4ms

Performance: Faster Timing

- Minitrace: **T**ime**S**tamp**C**ounter register (x86/x64) via *RDTSCP* instruction
  - √ 8 ns
  - √ Nanoseconds precision

Performance: Faster Timing

- Minitrace: **T**ime**S**tamp**C**ounter register (x86/x64) via *RDTSCP* instruction

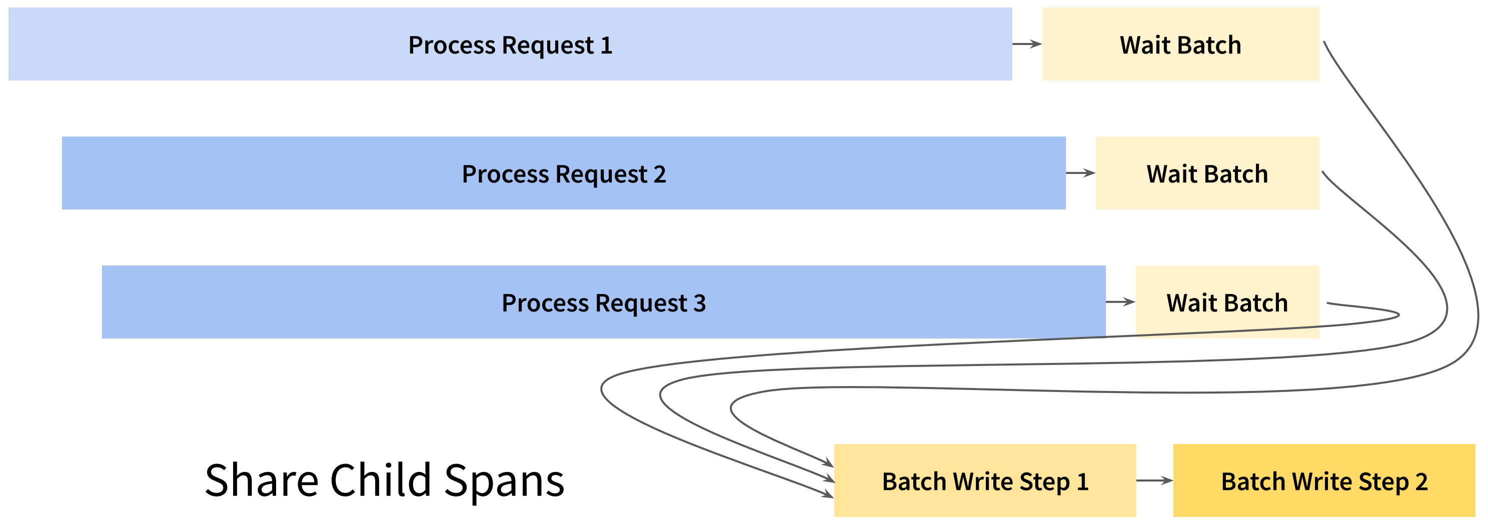
  √ 8 ns

  √ Nanoseconds precision
- **Caution:**
  - Without *CONSTANT_TSC* + *NONSTOP_TSC*, TSC is not synced in different cores
  - Even with these flags, TSC may be unsynced
    - VM, some CPU faults, …
  - Non x86: Fallback to CLOCK_MONOTONIC_COARSE

Performance: Serialization

- Memory spans ----(Serialization)---> Tracing Storage (e.g. Jaeger)
- **Complete** timing & collecting
- **Selective** reporting based on request latency

Trace Batch Systems



Share Child Spans

Community

- A **subset** of OpenTracing is implemented for performance.
- Built-in support: Spans can be reported to **Jaeger**.
- Early bird try out, you can use in your own projects:
  - https://github.com/tikv/minitrace-rust
- Some optimizations will be contributed to **opentelemetry-rust**.
  - We hope one day the official Rust client can adopt all optimizations!

TiKV & Resources

- Tracing will be available in the upcoming **TiKV v5.0**
- GitHub: https://github.com/tikv/tikv
- Website: https://tikv.org/
- Twitter: @tikvproject
- Slack: tikv-wg.slack.com